Refine Search

Search Results -

Terms	Documents
L32 and (first and second) near (data with base or database)	21

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search:

Refine Search

Recall Text | Clear | Interrupt

Search History

DATE: Wednesday, October 18, 2006 **Purge Queries** Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L33	L32 and (first and second) near (data with base or database)	21	L33
L32	L31 and (query$ or queryable or query) near cache	164	L32
L31	l30 and (internet or network or www or web) with servers	20496	L31
L30	707.clas.	38326	L30
L29	707/201	3467	L29
L28	L27 and peer-to-peer with cach$ near (network or internet or www)	5	L28
L27	709.clas.	48732	L27
L26	L25 and peer-to-peer with cach$ near (network or internet or www)	1	L26
L25	709/226	5041	L25
DB=PGPB; PLUR=YES; OP=OR			
L24	("20020065919")[PN]	1	L24
L23	("20020065919")[URPN]	0	L23
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L22	20020065919.pn.	2	L22

DB=USPT; PLUR=YES; OP=OR

L21	'5758342'.pn.	1	L21
L20	'5546582'.pn.	1	L20
L19	'6014667'.pn.	1	L19
L18	'5371886'.pn.	1	L18
L17	'5781910'.pn.	1	L17
L16	'5999931'.pn.	1	L16
L15	'6012059'.pn.	1	L15
L14	'6014667'.pn.	1	L14
L13	'6014669'.pn.	1	L13
L12	'6018745'.pn.	1	L12
L11	'6073141'.pn.	1	L11

DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR

L10	6243715.pn.	2	L10

DB=USPT; PLUR=YES; OP=OR

L9	'5212788'.pn.	1	L9
L8	'5212788'.pn.	1	L8
L7	'5261094'.pn.	1	L7
L6	'5261094'.pn.	1	L6
L5	'5434994'.pn.	1	L5
L4	'5434994'.pn.	1	L4
L3	'5613079'.pn.	1	L3
L2	'5625818'.pn.	1	L2

DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR

L1	5806074.pn.	2	L1

END OF SEARCH HISTORY

L33: Entry 15 of 21 File: USPT Nov 19, 2002

DOCUMENT-IDENTIFIER: US 6484161 B1
TITLE: Method and system for performing online data queries in a distributed computer system

Abstract Text (1):
Disclosed is a system for performing online data queries. The system for performing online data queries in a distributed computer system with a plurality of server nodes each fully redundant and capable of processing a user query request. Each server node includes a data query cache and other caches that may be used in performing data queries. The data query, as well as request allocation, is performed in accordance with an adaptive partitioning technique with a bias towards an initial partitioning scheme. Generic objects are created and used to represent business listings upon which the user may perform queries. Various data processing and integration techniques are included which enhance data queries. An update technique is used for synchronizing data updates as needed in updating the plurality of server nodes. A multi-media data transfer technique is used to transfer non-text or multi-media data between various components of the online query tool. Optimizations for searching, such as the common term optimization, are included for those commonly performed data queries. Also disclosed is a system for targeting advertisements that are displayed to a user of the system.

Brief Summary Text (7):
In accordance with principles of the invention is an apparatus for performing a data query. The apparatus includes a computer system having one or more server nodes and a database including the data used for performing the data query. A hardware router forwards a request including the data query to one of the server nodes. A backoffice component provides data included in the database. Each of the server nodes includes: a request router for determining if a request including a data query should be performed by each server node, or if the request should be routed elsewhere for processing; partitioning data used by the request router to determine where the request should be processed, the partitioning data associating each of the server nodes with a particular portion of a query domain upon which each server node primarily performs data queries; and a query cache including data associated with the particular portion of a query domain upon which each server node primarily performs data queries.

Drawing Description Text (32):
FIG. 33 is a flowchart of an example of method steps of one embodiment for performing data query cache lookup as used in performing a data query;

Drawing Description Text (34):
FIG. 35 is a flowchart of an embodiment of method with steps for forming a name and determining if the corresponding data set is located in the query cache;

Drawing Description Text (35):
FIG. 36 is an example of an entity as stored in the data query cache;

Detailed Description Text (3):
Referring now to FIG. 2, shown is an embodiment of a hardware view of an on-line query tool. In one embodiment, this on-line query tool may be the GTE

Superpages.SM. query tool. FIG. 2 shows a hardware view of the components that may be included in one embodiment of the query tool in typical operation as being accessed by a user through a network. The user 800 enters a query request which is sent via a network 802, such as the Internet, to the GTE Superpages Front End Server 804. The GTE Superpages Front End Server 804 includes a hardware router 806 for receiving incoming query requests. The hardware router routes the request, using a simple hardware-based technique, to one of the server nodes 808-810 which may be designated to service the request by performing the requested query. The servers 808 through 810, server 1 through server n, respectively, interact with the Primary Database 812 and Secondary Database 814 to perform a data query. The Primary Database 812 interacts with the Backoffice component 818 at times, as will be described in paragraphs elsewhere herein, to obtain data used in performing the queries. The Backoffice component 818 performs data filtering and other processing, for example, to combine information that may be obtained from various data sets producing a resultant data set. The resultant data set is subsequently transferred to the Primary Database for use by the various server nodes 808 through 810.

Detailed Description Text (10):
Referring now to FIG. 4, shown is an embodiment of the various software components for an on-line query system. One embodiment may be the on-line query tool of the GTE Superpages system. FIG. 4 depicts a software view of the typical operation of the system as being accessed by a user 800 through a network 802 using the hardware as described in conjunction with FIG. 2. As previously described, the user may enter a request, as through a browser. This request is communicated through the GTE Superpages Front End Server 804 over the network 802. As shown in FIG. 4, the Front End Server 804 includes server node 808 that includes a web server engine 852. In one embodiment, the web server engine 852 is a Netscape.TM. engine which serves as a central coordinating task for accessing files and displaying information to the user on the browser 824. The server node 808 also includes a request router 854, a monitor process 856 and a parser 866. The parser 866 generally includes a parse driver 858, a generic object dictionary 860, a query engine 862, and a data manager 864. The parse driver 858 operates upon data from a constructed ad repository 842 and the PHTML files 844. Additionally, the parse driver 858 stores and retrieves data from the PHTML execution tree 846 and the page cache 848. The data manager 864 included in the parser 866 is responsible for interacting with the database, which in the FIG. 4 is the Primary Database 812. It should also be noted that the data manager 864 may also obtain data from a Secondary Database as previously shown in FIG. 4. If there are multiple databases other than a Primary and Secondary Database, the data manager may also interact with these to obtain the necessary data upon which data queries are performed. The query engine 862 operates upon data from, and writes data to, the data query cache 850. Additionally, the query engine uses data from the term lists 836 to obtain identifiers and possibly other retrievable data in accordance with various key terms upon which a data query is being performed. The request router 854 generally interacts with the parser and reads data from the configuration file 830 and load file 834. The monitor process 856 also reads and writes data to and from respectively the load file 834. The web server engine 852, in this embodiment the Netscape engine 852, obtains data from the HTML repository 838 and the image repository 840 in accordance with various requests from the browser for different types of files. Each of the foregoing components will be described in more detail in terms of function and operation in paragraphs that follow. The monitor process 856 is generally responsible for indicating the availability of server nodes 808-810 in performing data queries. The monitor is also generally responsible for receiving incoming messages from other server nodes as to their availability for servicing requests.

Detailed Description Text (15):
In this particular embodiment, an incoming request may be processed by one of a plurality of parsers 858 on each of the server nodes. The parser 858 generally transforms the user input query into a form used by other components, such as the request router. The request router generally receives an incoming request as

forwarded by the hardware router 806 of FIG. 2. The request router subsequently uses the load file and the configuration file to decide which server node 808-810 a request is routed to based on the load and the availability of the server node, and the designated server for each partition or domain. Once a request is routed to one of the server nodes 808-810, the query is performed producing data query information that may be cached, for example, in the memory of a data query cache 850.

Detailed Description Text (16):
One use of the data query cache 850, as will be described in paragraphs that follow, is its use in improving the performance in response to a user request in a subsequent query that may use a subset or superset of the data stored in the data query cache 850. A superset or composition query is one which is a boolean composite of several querying terms. A composition query may be determined by the parser 866, and the request router 854 may decide to which server node 808-810 the composition query or other query is sent for processing in accordance with domain weights as indicated in the configuration file. Reallocation of requests when a server is unavailable may be performed generally with a bias toward the initial allocation scheme as indicated also by the configuration file. There is an assumption that reallocation of a request is on a transient basis, and that the initial allocation scheme is the one to be maintained. This concept will be described in paragraphs that follow in accordance with request routing and data query caching.

Detailed Description Text (19):
When processing an incoming user request which results in returning an HTML page to a user, a particular search order of the previously described caches and file systems may be performed. Initially, it is determined whether the HTML page to be displayed to the user is located in the page cache 848. If not, search results are obtained from the query cache and the resulting HTML page is constructed and itself may be placed in the page cache 848. If a PHTML file is required to be executed in constructing the resulting HTML file, the PHTML execution tree 846 may be accessed to determine if there is a parsed version of the required PHTML file already expanded in the PHTML execution tree. If no such file is located in the PHTML execution tree 846, the PHTML file 844 is accessed to obtain the required PHTML file. The order in which these caches and file systems are searched is generally in accordance with a graduated processing state of producing the resulting HTML file. Caches associated with a later state of processing are generally searched prior to ones associated with an earlier processing state in producing the resulting HTML file.

Detailed Description Text (50):
Referring back to FIG. 20, it may generally be noted that the Backoffice component 818 may include one or more database servers 892. A user may directly interact with the web server 894 included in the Backoffice component via connection 896 which, for example, may be a network connection of a user accessing the web server through the Internet. The user may also interact directly with the Backoffice component through the Front End Server Connection 822.

Detailed Description Text (54):
The query engine 862 is generally responsible for performing any required sorting of the query information or subsetting and supersetting of information. Generally, the query engine 862 retrieves various identifiers which act as keys into the Primary Database 812 or Secondary Database 814 for accessing particular pieces of information in response to a user query. After the query engine 862 formulates and retrieves various identifiers, for example as from the term lists, which correspond to a particular user query, this query information in the form of term list and retrieved information may be stored in the data query cache 850. A technique similar to the page cache query-to-filename mapping technique may be used to map a particular query request to a naming scheme by which data is accessed in the data

query cache. The technique for forming this name is described in other sections of this application.

Detailed Description Text (55):
Additionally, data which is stored in the data query cache 850 may be compressed or stored in a particular format which facilitates easy retrieval as well as attempting to optimize storage of the various data queries which are cached, as discussed in other portions of this application.

Detailed Description Text (61):
Referring now to FIG. 30, shown is a flowchart of the method steps of one embodiment for performing query engine processing. At step 962, the query engine receives an incoming request, as forwarded by the parse driver in step 954. At step 964, the data is retrieved for the "normal" search results as appropriate from the data query cache, or using an alternate technique. Details of this step are described in more detail in following paragraphs describing the use of the data query cache. Generally, "normal" search results refers to the resulting data set formed by business listing data associated with a well-defined geographic area. In addition to "normal" search result data are other search result data that may not be associated with a single well-defined geographic area, such as virtual businesses in the Internet. These other search results that may not be associated with a single well-defined geographic area are described in more detail in paragraphs relating to the data query cache and its use. At step 966, other search data in addition to the "normal" search data may be retrieved and integrated into the resulting data set. At step 968, the result data set is formulated in accordance with the user query request, such as displaying results in a particular order or beginning at a particular point. At step 970, the resulting data set is returned to the parse driver for formatting in a display format in an HTML file.

Detailed Description Text (77):
Referring to FIG. 6, a table 430 represents data stored in the generic object dictionary 860 corresponding to results of a search query provided by the query engine 862 or from the data query cache 850 in the case of a previous search having been performed. In the table 430, it is assumed that a search returns a plurality of objects corresponding to n categories and up to m listings for each of the categories. The annotation o.sub.jk means the object corresponding to the jth category and the kth listing. In the case of the table 430 (and thus the generic object dictionary 860), the objects may be object identifiers. For example, the field 412 may correspond to an object identifier of each of the objects 402, 404, 406. As discussed in more detail below, the parse driver 858 uses the table 430 provided by the generic object dictionary 860 along with the PHTML execution trees 846, to provide specific HTML code from the parse driver 858 to the browser 824 of the user 802.

Detailed Description Text (80):
For each query that is presented to the query engine 862, the query engine 862 determines whether the query is found in the data query cache 850 or whether it is necessary to perform a query operation using the Verity software (discussed elsewhere herein) and the term list 836. In either instance, the results of the query are provided by the query engine 862 to the generic object dictionary 860 in a form set forth above in connection with the description of FIG. 6. The parse driver 858 and PHTML execution trees 846 then operate on the generic object dictionary 860 to determine what data is displayed to the user by the browser 824. In some instances, the PHTML execution trees 846 may require the parse driver 858 to obtain additional data from the databases 812, 814 through the data manager 864. For example, in instances where the categories corresponding to the retrieved documents (business listings) are displayed, the PHTML execution trees 846 may cause the parse driver 858 to obtain information from the generic object dictionary 860 that identifies each category and the number of listing corresponding to each category. Then, the portion of the PHTML execution trees 846 may cause the parse

driver 858 to use the data manager 864 to access additional information from the databases 812, 814, such as the names of the categories corresponding to the category identifiers provided in the generic object dictionary 860.

Detailed Description Text (88):
When performing the routing of particular requests, such as data queries, existing systems may perform request routing to a particular server in a distributed computer system without reference to certain available factors, such as an initial partitioning of the entire domain, or an assumption that data queries will be cached in a data query cache and subsequently reused for additional searches. Generally, using the concepts which will be described in paragraphs that follow, the larger the number of queries that are performed when routed to a particular node in accordance with an initial allocation scheme, the quicker subsequent searches on this same particular node may be performed due to the use of the data query cache.

Detailed Description Text (94):
Consider the following example of the use of the data query cache and subsequent searches which use a subset of the data stored in the cache. For example, suppose the first request results in a query of all of the restaurants within thirty (30) miles of Boston. This query data is placed in the data query cache. A second request results in a query of all the seafood restaurants within thirty (30) miles of Boston. The second request is routed to the same node as the first request in accordance with loading configuration files, for example, as shown on FIG. 4. The second query is performed quickly by using the data query cache information and searching for a subset of the cached data indicating restaurants within thirty (30) miles of Boston for a subset of this first search data which indicates seafood restaurants. Subsequently, this second request query data which indicated all the seafood restaurants within thirty (30) miles of Boston is also stored as a separate data set within the data query cache.

Detailed Description Text (95):
It should generally be noted that the data included in the data query cache is placed in nonvolatile storage such that if the node were to become unavailable, data from the data cache may be fully restored once the node resumes service.

Detailed Description Text (96):
The composition query also uses the data in the data query cache. A composition query may generally be referred to as one which is a composition of several queries, for example, when using several conjunctive search terms. For example, a request of all the French restaurants in Massachusetts, Texas and California is a composition query that may reuse any existing cached data from previous queries stored individually regarding restaurants in Massachusetts, Texas and California. A composition query is generally determined by the Parse Driver, and the request router decides to which server node 808-810 within the Front End Server the composition query is sent for processing in accordance with domain weights of the configuration file. Consider the following Configuration File information based upon the previous composition query:

Detailed Description Text (98):
In the above caching scheme, a particular domain is associated with a particular server node upon which data query caching is performed for designated domains. The domain and server weights reflect the cost associated with processing a request on each node using the data query cache. Accordingly, routing a request in accordance with these weights results in faster subsequent query times for those requests.

Detailed Description Text (100):
At a first time, node M1 becomes unavailable and the routers reallocate Domain D1 to node M2 and D2 to node M3. At a second time, node M2 also becomes unavailable. Domains D1 and D3 are reallocated to node M3 in addition to domains D5 and D6.

Domain D4 is reallocated to node M4 in addition to domains D7 and D8. At a third time, node M1 is restored and node M2 is still unavailable. Domains D1 and D2 are reallocated to M1 in addition to Domain D3. Domains D5, D6 and D4 are allocated to node M3. Domains D7 and D8 are allocated to node M4. There is a bias toward restoring the initial allocation scheme when a node becomes available. This bias contributes to faster subsequent query times upon re-entry of a server node due to the use of the data query cache, and routing of subsequent requests to the particular nodes in accordance with this bias.

Detailed Description Text (102):
Referring now to FIG. 33, shown is an example embodiment of a flowchart of method steps for performing a data query. At step 200, a determination is made as to whether a data set in the data query cache corresponds to the current query being made. If so, control proceeds to step 202 where this data is retrieved and used by the query engine in formulating the query results that are displayed to the user. At this point, the processing stops at step 216.

Detailed Description Text (103):
If a determination is made at step 200 that no data set in the data query cache corresponds to the current query being made, control proceeds to step 204 where parents of the data query are determined. In this embodiment, parents of the current query are determined by dropping one of the terms. For example, if the query being made is for "MA AND RESTAURANTS AND FLOWERSHOPS", each of the three terms is sequentially dropped to form all combinations of two possible terms. In this instance, the set of parents is the following: MA AND RESTAURANTS MA AND FLOWERSHOPS RESTAURANTS AND FLOWERSHOPS

Detailed Description Text (104):
It should be generally noted that in this embodiment, a search is made for only the parent terms. Similarly, other embodiments may go further in searching for results in the data query cache by also forming grandparent terms, as by dropping two terms. This process can be repeated for any number of terms being dropped and subsequently determining if any data sets in the data query cache correspond to the resulting terms.

Detailed Description Text (105):
At step 205, a determination is made as to whether data results in the data query cache correspond to any of the parent terms. If not, control proceeds to step 212 where a closest ancestor may be used as a basis for starting to form the resulting data set. In one embodiment, preprocessing insures that ancestor-based geography exists. In one implementation, that ancestor is a Verity term list associated with a particular state. This implementation uses API calls to retrieve the data identifiers corresponding to the resulting data to be included in the query results.

Detailed Description Text (106):
If, at step 205, it is determined that there are one or more data sets in the data query cache that correspond to one or more of the parent terms, control proceeds to step 206 where a cost is associated with each parent. One embodiment associates a cost with each parent term in accordance with the number of listings of each parent term. This may also be normalized and used in a percentage form by dividing the number of listings in the parent domain by the total number of listings in the query domain. This percentage represents the probability of a business listing belonging to the parent data set appearing in the database. Control proceeds to step 208 where the parent with the minimum cost is chosen as the starting data set for formulating the data results. At step 210, the minimum cost derivation sequence is applied to produce the resulting data query. Generally, the minimum cost derivation sequence is obtained by operating upon the least probability terms first.

Detailed Description Text (108):
Referring now to FIG. 34, shown is a diagram of one example used in step 210 for determining and applying the best derivation sequence. In this example, the query is for MA AND RESTAURANTS AND FLOWERSHOPS. As represented in state 230, it has been determined that MA is the starting data set which is located in the data query cache. In this example, the parentage has been extended to grandparents, and MA has been determined to be the first ranking data set in terms of parentage and number of listings in the data set. At this point, control proceeds to one of two states, 232 representing "MA AND RESTAURANTS", or 234 representing "MA AND FLOWERSHOPS". The state to which control is advanced depends generally on choosing the path with the minimum associated cost at each step. In this instance, the number of elements in the data sets "FLOWERSHOPS" (state 234) and "RESTAURANTS" (state 232) may be considered in determining cost. If the number of elements in FLOWERSHOPS is less than the number of elements in the data set RESTAURANTS, control proceeds to state 234 where each business listing in the data set FLOWERSHOP is examined to determine if it is also in MA. The resulting data set forms the set of all business listings in MA AND FLOWERSHOPS. In contrast, if the number of elements in the data set RESTAURANTS is less than FLOWERSHOPS, state 232 is entered and similar searching of the data set is performed. From either state 232 or 234, control proceeds to state 236 where searching of the data set elements is performed to produce the final resulting data set representing "MA AND RESTAURANTS AND FLOWERSHOPS". Generally, the approach just described is to advance to the next state which has the minimum cost associated until the final resulting data set is determined.

Detailed Description Text (109):
It should also be noted that some of the determination of data sets as used in performing queries may be done as preprocessing to partition the data sets. For example, in one embodiment, the data is partitioned by states. The adaptive techniques as described with regard to the GTE Superpages application described herein include partitioning the data sets based on geography, particularly within each state. In this instance, particular server nodes are designated as primary query servers based on geographic location by state. Additionally, as part of this partitioning of requests, the data query caches and term lists of identifiers are also partitioned according to state. In this embodiment, this partitioning is done as a preprocessing step prior to servicing a request in that the identifiers are formed and placed on each dedicated server node. Similarly, other data partitioning may also be performed as part of a preprocessing step. Generally, this partitioning may be determined based on expected data queries and data sets formed accordingly, for example, by examining log files with recorded data query search histories to determine frequently searched categories or combinations of categories.

Detailed Description Text (113):
The Data Query Cache 850, in this embodiment, generally includes a "hot" and "cold" cache. In this embodiment, the caching technique implemented is the LRU (Least Recently Used) policy by which elements of the cache are selected for replacement in accordance with time from last use. These and other policies are generally known to those skilled in the art. Generally, the "hot" cache may include the most recently used items and the cold cache the remaining items. In this embodiment, each of the data query caches and other caching elements as depicted in FIG. 2, may be fast memory access devices, as known to those skilled in the art, used generally for caching.

Detailed Description Text (116):
Data sets that are stored in the data query cache and page cache each correspond to a particular search query. In other words, a mapping technique may be used to map a particular query to corresponding data as stored in the data query cache and the page cache. Generally, this mapping uniquely maps a data query to a name referring to the data set of the data query. In this embodiment, this allows quick access of the data set associated with a particular query and quick determination if such a data set exists, for example, in the data query cache.

Detailed Description Text (117):
Referring now to FIG. 35, shown is a flowchart of an embodiment of the steps for forming a name associated with a data set, as may be stored in the data query cache or page cache. At step 240, a subset of query terms is determined such that a string representing a particular query is uniquely mapped to a name corresponding to a data set. In this embodiment, the subset of keys that are used in mapping a string corresponding to a query to a name of a data set include: Proximity, City, State, Street, Zip, Category, Category Identifier, Business name, Area code, Phone number, Keywords, and National Account.

Detailed Description Text (119):
At step 244, a query string corresponding to a particular user query is formed using the original string as formed, for example, by the Parser of FIG. 2. The query string includes only those terms which are included in the subset as identified in step 240. If the original string does not include an item that is in the subset, for example, since the user query does not include the item as a search term, that item is omitted in forming the query string corresponding to the data set. At step 248, this query string is used to determine if a data set is located in the data query cache that corresponds to the current user query request. In this embodiment, the data sets each correspond to a filename. Thus, a lookup as to whether a data set corresponding to a particular user query exists may be determined by performing a directory lookup, for example, using file system services as may be included in an operating system upon a device which serves as a fast memory access or other caching device.

Detailed Description Text (120):
It should be noted that this technique may be used generally within the Superpages Front End Server and Backoffice to form unique names that correspond to particular search terms. For example, one embodiment may include services for operating upon the original query string as formed by the Parser to produce parents and grandparents of the terms included in a query when performing the method steps of FIGS. 33 and 34 if there is no exact data set match in the data query cache. This may provide the advantage of insulating other code, such as in data encapsulation, from knowing the internal structure of the query string. Generally, as known to those skilled in the art, this is a common programming technique to minimize code portions from changes in data types and structures to minimize, for example, the amount of recompilation when a new data type is introduced or existing data type modified. Other techniques, such as hashing, may be used to generate a unique identifier for the input string, as known to those skilled in the art.

Detailed Description Text (121):
It should be generally noted that a similar mapping technique is used in forming a Page Cache name. The technique used is as described for forming the Query Cache filename with additional qualifying terms in accordance with the "look and feel", such as display features, used to produce the Page Cache name. For example, if the displayed resulting HTML page includes 15 listings/page, the Page Cache name includes a parameter in forming the name uniquely identifying the filename including the result set for a query in this particular display format.

Detailed Description Text (122):
Generally, in this embodiment, the data query cache includes cache objects in which each cache object corresponds to a particular cached query resulting data set. Referring now to FIG. 36, shown is a block diagram of one embodiment of a data set as stored in the data query cache. Generally, each data set 250 includes header information 252 and information corresponding to one or more business listings. Generally, header information may include information describing the data query set, such as the number of business listings in the data set. Other types of information may be included in accordance with each particular application and implementation.

Detailed Description Text (125):
The technique used to store the data in the data cache from memory includes object serialization and deserialization techniques, as known to those of ordinary skill in the art. These techniques transform an internal storage format, as may be stored in random access memory, to a format suitable for persistent storage in a file system, as in the data query cache. The complementary operation is also performed from persistent storage to the in-memory copy. For each of the above-named fields, object serialization, i.e., from memory to persistent storage device in cache, is performed by storing the data type, its length, and the data itself. It should be noted that the length may not be needed for each data field, for example, in fixed length data types. The complementary operation of object deserialization is generally performed by reading the fields in the same order as written to the cache.

Detailed Description Text (127):
It should be noted that in this particular embodiment, the data query cache may include different types of cached geographical data as may be used in performing different data queries. For example, the type of data cached described in the prior paragraphs is the "normal" business listing data as associated with a well-defined geographic area. Other businesses, for example, such as a florist or an airline, may not be associated with a single well-defined geographic location. A business may not have any geographic bounds, such as if it is an Internet business with a virtual storefront accessible on the Internet. Also, other businesses may be located in a particular well-defined geographic area, such as an airline with a physical presence in a particular city, but the service area which corresponds to the service offered does not correspond to the location of the business itself. To include businesses with these particularities, in addition to the "normal" business listing just described in which the geographic business location and service areas correspond, the concepts of multi-city and total-city placements have been included in this embodiment.

Detailed Description Text (132):
These key-value pairs are extracted from the original query string and appended to the "SCOPE=T" to form the total-city cache name. In one embodiment, these functions of extracting the information from the original query string and forming the total-city cache name may be performed by the same software as forming the name for the data query cache "normal" query name, such as by API calls to the same routines with parameters, as known to those of ordinary skill in the art of programming.

Detailed Description Text (138):
In all the caches, a garbage collection technique may be included to remove or delete cached objects that have been determined to be "old" in accordance with predetermined criteria. For example, in one embodiment using the LRU caching scheme, whenever the amount of free cache space falls below a threshold level, the garbage collection routine is invoked. The threshold level includes parameters relating to a predetermined number of cache objects and the accumulated size of the objects in the cache. In this embodiment, although there may be multiple conceptual caches, such as the "normal" data query cache, the multi-city cache, and the total-city cache, the cached results may physically reside in the same "hot" and "cold" caching devices. However, in this embodiment, the different types of caching results may be accessed independent of the other caching results. Other embodiments may have other organizations of the caches in accordance with other implementation and associated data requirements.

Detailed Description Text (164):
Referring now to FIG. 45, shown is one embodiment of the database included in the Backoffice component as included in FIGS. 2 and 4. Generally, data updates included in the database come from three different sources in this particular embodiment. One source is on-line updates, as provided by users making updates or entering new

information for business listing via network connections through the Backoffice component as through the Front End Server. A second source of data updates is based on foreign source updates. Generally, foreign source updates are those update records which come from a different data source than the original existing database. A third type of data integration or update source is referred to as a native source update. Generally, a native source update is when an updated version of the existing database having the same source as the existing database is provided. For example, a database copy may be provided as an update on a monthly basis using full sets of data where a data provider provides an updated version of the same data set. The native source data integration procedure integrates those changes in the new data set into the existing database. This is in contrast to a foreign source update, for example, where the existing database is provided by one vendor, and the update records for example, are provided by a different vendor. The update vendors being from a foreign source are called foreign source data integration or updates.

Detailed Description Text (221):
Generally, the text data included in this data transfer may be characterized as structured data, as included in text which is displayed to the user. The second type of data generally transferred is denoted as "blob" data which is generally not able to be decomposed or operated upon in different portions. For example, blob data may include a machine-executable program which is generally binary data type. Generally, the technique uses two separate data channels in which each channel transfers a different type of data. In this particular embodiment, one data channel is used to transfer the text data, and Database Link.TM. software, as included in the commercially available Oracle.TM. database, is used to facilitate database communication of text data. Therefore the database routines, such as those included in the Database Link software, may be used in transferring text data between databases. In this particular embodiment, the Oracle database does not support direct non-text manipulation, such as for transferring data of different types, such as blob data. Therefore, a second different data channel is used to transfer the blob data from one database to another in which the second channel is external to the database since the version of the Oracle database software used in this embodiment does not provide the needed support for direct non-text data manipulation. The blob data, which may also generally be characterized as multi-media data, is transferred asynchronously from the text data between databases.

Detailed Description Text (242):
The overall technique is generally to copy the text and blob or multi-media data asynchronously on two separate channels. This data is copied from a first database to a second database. Initially, the data is located on the second database in a temporary location until all of the portions of the data associated with a particular data transfer arrive at the second database. When it has been determined that all portions of the data have successfully arrived on the second database, the assembly process of copying the data from the temporary locations and merging the information into other data tables is performed on the second database.

Detailed Description Text (244):
The data transfer technique described is generally a technique for transferring data using two data links between two databases. One of these data links is an internal data link with respect to the database, the second data link is an external data link with respect to the database. The internal data link is optimized for the structured text data transfer while the external one is optimized for the multi-media data transfer, such as the transference of data stored in binary objects in the database. This technique for data transfer generally alleviates the limitations of the existing database technology which does not provide for the transferring of multi-media objects using the internal data link. Moreover, by using the two data links to transfer the various data types, performance and stability are improved over an alternative prior art approach which uses only the external link for transferring both text and multi-media or blob

data.

Detailed Description Text (253):
Once the data modifications are incorporated into the Backoffice component, the data updates, including the updates to advertisement data and other data associated with each business listing, may be propagated to the Front End Server component. The non-text or multi-media data, for example, as included in advertisements with image files, may be transferred to the Front End Server from the Backoffice using multi-media transfer techniques, as generally described in other sections of this description. The updates to the Primary Database included in the Front End Server may be communicated as a table of commands created in the Backoffice component and transferred, as by a network connection, to the Front End Server. Generally, in this embodiment, the table created in the Backoffice includes an application developed command language corresponding to the various types of record updates and modifications that may be included in this particular embodiment. Each of these commands may be further translated in the Front End Server into one or more actual database commands that perform the table operation. For example, an entry in the table of database update commands may be specified as follows: COMMAND RECORD# OPTIONAL DATA DELETE 1-5

Detailed Description Text (270):
Also, in this particular embodiment, the contents of the Page Cache 848 and the Query Cache 850 are reinitialized when an update is performed, as in performing the incremental update procedures described in conjunction with FIGS. 31 and 32. The data included in the PHTML execution tree is also reinitialized.

Current US Class (1):
707

CLAIMS:

1. An apparatus for performing a data query comprising: a computer system having one or more server nodes and a database including data used in connection with performing the data query; a hardware router for forwarding a request including the data query to one of said server nodes; a backoffice component for providing data included in the database; and wherein each of said server nodes includes: a request router for determining if a request including a data query should be performed by said each server node; partitioning data used by said request router to determine which of said one or more server nodes should process said request, said partitioning data associating each of said one or more server nodes with a particular portion of a query domain upon which said each server node primarily performs data queries, and said partitioning data including a static file having weighted parameters of the query domain upon which each said server node primarily performs data queries; and a query cache including data associated with said particular portion of a query domain upon which said each server node primarily performs data queries.

5. The apparatus of claim 1, wherein said one or more server nodes communicate using a network.

22. A method executed in a computer system for performing a data query comprising: forwarding, by a hardware router, a request including the data query to a first of one or more server nodes included in said computer system; forming partitioning data that associates each of said one or more server nodes with a particular portion of a query domain upon which said each server node primarily performs data queries using data stored in a database, and said partitioning data including a static file having weighted parameters of the query domain upon which each said server node primarily performs data queries; determining, by a request router using said partitioning data, if said request including said data query should be performed by said first server node, or if said request should be routed to another

of said one or more server nodes for processing; and forming a result data set in accordance with said data query using a query cache, said query cache including data associated with said particular portion of said query domain upon which said first server node primarily performs data queries.

26. The method of claim 22, wherein said one or more server nodes communicate using a network.

43. A computer program product for performing a data query comprising: means for forwarding, by a hardware router, a request including the data query to a first of one or more server nodes; means for forming partitioning data that associates each of said one or more server nodes with a particular portion of a query domain upon which said each server node primarily performs data queries using data stored in a database, said partitioning data including a static file having weighted parameters of the query domain upon which each said server node primarily performs data queries; means for determining, by a request router using said partitioning data, if said request including said data query should be performed by said first server node, or if said request should be routed to another of said one or more server nodes for processing; and means for forming a result data set in accordance with said data query using a query cache, said query cache including data associated with said particular portion of said query domain upon which said first server node primarily performs data queries.

45. An apparatus for performing a data query comprising: a computer system having one or more server nodes and a database including data used in connection with performing the data query; a hardware router for forwarding a request including the data query to one of said server nodes; a backoffice component for providing data included in the database; and wherein each of said server nodes includes: a request router for determining if a request including a data query should be performed by said each server node; partitioning data used by said request router to determine which of said one or more server nodes should process said request, said partitioning data associating each of said one or more server nodes with a particular portion of a query domain upon which said each server node primarily performs data queries; wherein said partitioning data is disjoint such that said query domain is disjointly partitioned among said one or more server nodes designating a particular one of said server nodes as being a primary server for performing data queries related to an associated partition of said query domain; and a query cache including data associated with said particular portion of a query domain upon which said each server node primarily performs data queries.

46. A method executed in a computer system for performing a data query comprising: forwarding, by a hardware router, a request including the data query to a first of one or more server nodes included in said computer system; forming partitioning data that associates each of said one or more server nodes with a particular portion of a query domain upon which said each server node primarily performs data queries using data stored in a database; wherein said partitioning data includes, for each of said one or more server nodes: said particular portion of said query domain associated with said each server node; a domain weight representing costs associated with processing a request for said particular portion of said query domain; a server weight representing a cost associated with processing a request on said each server node; and an availability status of said each server node; determining, by a request router using said partitioning data, if said request including said data query should be performed by said first server node, or if said request should be routed to another of said one or more server nodes for processing; and forming a result data set in accordance with said data query using a query cache, said query cache including data associated with said particular portion of said query domain upon which said first server node primarily performs data queries.

Previous Doc Next Doc Go to Doc#

L33: Entry 7 of 21 File: USPT Nov 30, 2004

DOCUMENT-IDENTIFIER: US 6826559 B1
TITLE: Hybrid category mapping for on-line query tool

Abstract Text (1):
Disclosed is a system for performing online data queries. The system for performing online data queries is a distributed computer system with a plurality of server nodes each fully redundant and capable of processing a user query request. Each server node includes a data query cache and other caches that may be used in performing data queries. The data query, as well as request allocation, is performed in accordance with an adaptive partitioning technique with a bias towards an initial partitioning scheme. Generic objects are created and used to represent business listings upon which the user may perform queries. Various data processing and integration techniques are included which enhance data queries. An update technique is used for synchronizing data updates as needed in updating the plurality of server nodes. A multi-media data transfer technique is used to transfer non-text or multi-media data between various components of the online query tool. Optimizations for searching, such as the common term optimization, are included for those commonly performed data queries. Also disclosed is a system for targeting advertisements that are displayed to a user of the system.

Drawing Description Text (32):
FIG. 33 is a flowchart of an example of method steps of one embodiment for performing data query cache lookup as used in performing a data query;

Drawing Description Text (34):
FIG. 35 is a flowchart of an embodiment of method with steps for forming a name and determining if the corresponding data set is located in the query cache;

Drawing Description Text (35):
FIG. 36 is an example of an entity as stored in the data query cache;

Detailed Description Text (3):
Referring now to FIG. 2, shown is an embodiment of a hardware view of an on-line query tool. In one embodiment, this on-line query tool may be the GTE Superpages.SM. query tool. FIG. 2 shows a hardware view of the components that may be included in one embodiment of the query tool in typical operation as being accessed by a user through a network. The user 800 enters a query request which is sent via a network 802, such as the Internet, to the GTE Superpages Front End Server 804. The GTE Superpages Front End Server 804 includes a hardware router 806 for receiving incoming query requests. The hardware router routes the request, using a simple hardware-based technique, to one of the server nodes 808-810 which may be designated to service the request by performing the requested query. The servers 808 through 810, server 1 through server n, respectively, interact with the Primary Database 812 and Secondary Database 814 to perform a data query. The Primary Database 812 interacts with the Backoffice component 818 at times, as will be described in paragraphs elsewhere herein, to obtain data used in performing the queries. The Backoffice component 818 performs data filtering and other processing, for example, to combine information that may be obtained from various data sets producing a resultant data set. The resultant data set is subsequently transferred to the Primary Database for use by the various server nodes 808 through 810.

Detailed Description Text (10):
Referring now to FIG. 4, shown is an embodiment of the various software components for an on-line query system. One embodiment may be the on-line query tool of the GTE Superpages system. FIG. 4 depicts a software view of the typical operation of the system as being accessed by a user 800 through a network 802 using the hardware as described in conjunction with FIG. 2. As previously described, the user may enter a request, as through a browser. This request is communicated through the GTE Superpages Front End Server 804 over the network 802. As shown in FIG. 4, the Front End Server 804 includes server node 808 that includes a web server engine 852. In one embodiment, the web server engine 852 is a Netscape.TM. engine which serves as a central coordinating task for accessing files and displaying information to the user on the browser 824. The server node 808 also includes a request router 854, a monitor process 856 and a parser 866. The parser 866 generally includes a parse driver 858, a generic object dictionary 860, a query engine 862, and a data manager 864. The parse driver 858 operates upon data from a constructed ad repository 842 and the PHTML files 844. Additionally, the parse driver 858 stores and retrieves data from the PHTML execution tree 846 and the page cache 848. The data manager 864 included in the parser 866 is responsible for interacting with the database, which in the FIG. 4 is the Primary Database 812. It should also be noted that the data manager 864 may also obtain data from a Secondary Database as previously shown in FIG. 4. If there are multiple databases other than a Primary and Secondary Database, the data manager may also interact with these to obtain the necessary data upon which data queries are performed. The query engine 862 operates upon data from, and writes data to, the data query cache 850. Additionally, the query engine uses data from the term lists 836 to obtain identifiers and possibly other retrievable data in accordance with various key terms upon which a data query is being performed. The request router 854 generally interacts with the parser and reads data from the configuration file 830 and load file 834. The monitor process 856 also reads and writes data to and from respectively the load file 834. The web server engine 852, in this embodiment the Netscape engine 852, obtains data from the HTML repository 838 and the image repository 840 in accordance with various requests from the browser for different types of files. Each of the foregoing components will be described in more detail in terms of function and operation in paragraphs that follow. The monitor process 856 is generally responsible for indicating the availability of server nodes 808-810 in performing data queries. The monitor is also generally responsible for receiving incoming messages from other server nodes as to their availability for servicing requests.

Detailed Description Text (15):
In this particular embodiment, an incoming request may be processed by one of a plurality of parsers 858 on each of the server nodes. The parser 858 generally transforms the user input query into a form used by other components, such as the request router. The request router generally receives an incoming request as forwarded by the hardware router 806 of FIG. 2. The request router subsequently uses the load file and the configuration file to decide which server node 808-810 a request is routed to based on the load and the availability of the server node, and the designated server for each partition or domain. Once a request is routed to one of the server nodes 808-810, the query is performed producing data query information that may be cached, for example, in the memory of a data query cache 850.

Detailed Description Text (16):
One use of the data query cache 850, as will be described in paragraphs that follow, is its use in improving the performance in response to a user request in a subsequent query that may use a subset or superset of the data stored in the data query cache 850. A superset or composition query is one which is a boolean composite of several querying terms. A composition query may be determined by the parser 866, and the request router 854 may decide to which server node 808-810 the composition query or other query is sent for processing in accordance with domain

weights as indicated in the configuration file. Reallocation of requests when a server is unavailable may be performed generally with a bias toward the initial allocation scheme as indicated also by the configuration file. There is an assumption that reallocation of a request is on a transient basis, and that the initial allocation scheme is the one to be maintained. This concept will be described in paragraphs that follow in accordance with request routing and data query caching.

Detailed Description Text (19):
When processing an incoming user request which results in returning an HTML page to a user, a particular search order of the previously described caches and file systems may be performed. Initially, it is determined whether the HTML page to be displayed to the user is located in the page cache 848. If not, search results are obtained from the query cache and the resulting HTML page is constructed and itself may be placed in the page cache 848. If a PHTML file is required to be executed in constructing the resulting HTML file, the PHTML execution tree 846 may be accessed to determine if there is a parsed version of the required PHTML file already expanded in the PHTML execution tree. If no such file is located in the PHTML execution tree 846, the PHTML file 844 is accessed to obtain the required PHTML file. The order in which these caches and file systems are searched is generally in accordance with a graduated processing state of producing the resulting HTML file. Caches associated with a later state of processing are generally searched prior to ones associated with an earlier processing state in producing the resulting HTML file.

Detailed Description Text (50):
Referring back to FIG. 20, it may generally be noted that the Backoffice component 818 may include one or more database servers 892. A user may directly interact with the web server 894 included in the Backoffice component via connection 896 which, for example, may be a network connection of a user accessing the web server through the Internet. The user may also interact directly with the Backoffice component through the Front End Server Connection 822.

Detailed Description Text (54):
The query engine 862 is generally responsible for performing any required sorting of the query information or subsetting and supersetting of information. Generally, the query engine 862 retrieves various identifiers which act as keys into the Primary Database 812 or Secondary Database 814 for accessing particular pieces of information in response to a user query. After the query engine 862 formulates and retrieves various identifiers, for example as from the term lists, which correspond to a particular user query, this query information in the form of term list and retrieved information may be stored in the data query cache 850. A technique similar to the page cache query-to-filename mapping technique may be used to map a particular query request to a naming scheme by which data is accessed in the data query cache. The technique for forming this name is described in other sections of this application.

Detailed Description Text (55):
Additionally, data which is stored in the data query cache 850 may be compressed or stored in a particular format which facilitates easy retrieval as well as attempting to optimize storage of the various data queries which are cached, as discussed in other portions of this application.

Detailed Description Text (61):
Referring now to FIG. 30, shown is a flowchart of the method steps of one embodiment for performing query engine processing. At step 962, the query engine receives an incoming request, as forwarded by the parse driver in step 954. At step 964, the data is retrieved for the "normal" search results as appropriate from the data query cache, or using an alternate technique. Details of this step are described in more detail in following paragraphs describing the use of the data

query cache. Generally, "normal" search results refers to the resulting data set formed by business listing data associated with a well-defined geographic area. In addition to "normal" search result data are other search result data that may not be associated with a single well-defined geographic area, such as virtual businesses in the Internet. These other search results that may not be associated with a single well-defined geographic area are described in more detail in paragraphs relating to the data query cache and its use. At step 966, other search data in addition to the "normal" search data may be retrieved and integrated into the resulting data set. At step 968, the result data set is formulated in accordance with the user query request, such as displaying results in a particular order or beginning at a particular point. At step 970, the resulting data set is returned to the parse driver for formatting in a display format in an HTML file.

Detailed Description Text (77):
Referring to FIG. 6, a table 430 represents data stored in the generic object dictionary 860 corresponding to results of a search query provided by the query engine 862 or from the data query cache 850 in the case of a previous search having been performed. In the table 430, it is assumed that a search returns a plurality of objects corresponding to n categories and up to m listings for each of the categories. The annotation o.sub.jk means the object corresponding to the jth category and the kth listing. In the case of the table 430 (and thus the generic object dictionary 860), the objects may be object identifiers. For example, the field 412 may correspond to an object identifier of each of the objects 402, 404, 406. As discussed in more detail below, the parse driver 858 uses the table 430 provided by the generic object dictionary 860 along with the PHTML execution trees 846, to provide specific HTML code from the parse driver 858 to the browser 824 of the user 802.

Detailed Description Text (80):
For each query that is presented to the query engine 862, the query engine 862 determines whether the query is found in the data query cache 850 or whether it is necessary to perform a query operation using the Verity software (discussed elsewhere herein) and the term list 836. In either instance, the results of the query are provided by the query engine 862 to the generic object dictionary 860 in a form set forth above in connection with the description of FIG. 6. The parse driver 858 and PHTML execution trees 846 then operate on the generic object dictionary 860 to determine what data is displayed to the user by the browser 824. In some instances, the PHTML execution trees 846 may require the parse driver 858 to obtain additional data from the databases 812, 814 through the data manager 864. For example, in instances where the categories corresponding to the retrieved documents (business listings) are displayed, the PHTML execution trees 846 may cause the parse driver 858 to obtain information from the generic object dictionary 860 that identifies each category and the number of listing corresponding to each category. Then, the portion of the PHTML execution trees 846 may cause the parse driver 858 to use the data manager 864 to access additional information from the databases 812, 814, such as the names of the categories corresponding to the category identifiers provided in the generic object dictionary 860.

Detailed Description Text (88):
When performing the routing of particular requests, such as data queries, existing systems may perform request routing to a particular server in a distributed computer system without reference to certain available factors, such as an initial partitioning of the entire domain, or an assumption that data queries will be cached in a data query cache and subsequently reused for additional searches. Generally, using the concepts which will be described in paragraphs that follow, the larger the number of queries that are performed when routed to a particular node in accordance with an initial allocation scheme, the quicker subsequent searches on this same particular node may be performed due to the use of the data query cache.

Detailed Description Text (94):
Consider the following example of the use of the data query cache and subsequent searches which use a subset of the data stored in the cache. For example, suppose the first request results in a query of all of the restaurants within thirty (30) miles of Boston. This query data is placed in the data query cache. A second request results in a query of all the seafood restaurants within thirty (30) miles of Boston. The second request is routed to the same node as the first request in accordance with loading configuration files, for example, as shown on FIG. 4. The second query is performed quickly by using the data query cache information and searching for a subset of the cached data indicating restaurants within thirty (30) miles of Boston for a subset of this first search data which indicates seafood restaurants. Subsequently, this second request query data which indicated all the seafood restaurants within thirty (30) miles of Boston is also stored as a separate data set within the data query cache.

Detailed Description Text (95):
It should generally be noted that the data included in the data query cache is placed in nonvolatile storage such that if the node were to become unavailable, data from the data cache may be fully restored once the node resumes service.

Detailed Description Text (96):
The composition query also uses the data in the data query cache. A composition query may generally be referred to as one which is a composition of several queries, for example, when using several conjunctive search terms. For example, a request of all the French restaurants in Massachusetts, Texas and California is a composition query that may reuse any existing cached data from previous queries stored individually regarding restaurants in Massachusetts, Texas and California. A composition query is generally determined by the Parse Driver, and the request router decides to which server node 808-810 within the Front End Server the composition query is sent for processing in accordance with domain weights of the configuration file.

Detailed Description Text (99):
In the above caching scheme, a particular domain is associated with a particular server node upon which data query caching is performed for designated domains. The domain and server weights reflect the cost associated with processing a request on each node using the data query cache. Accordingly, routing a request in accordance with these weights results in faster subsequent query times for those requests.

Detailed Description Text (105):
At a first time, node M1 becomes unavailable and the routers reallocate Domain D1 to node M2 and D2 to node M3. At a second time, node M2 also becomes unavailable. Domains D1 and D3 are reallocated to node M3 in addition to domains D5 and D6. Domain D4 is reallocated to node M4 in addition to domains D7 and D8. At a third time, node M1 is restored and node M2 is still unavailable. Domains D1 and D2 are reallocated to M1 in addition to Domain D3. Domains D5, D6 and D4 are allocated to node M3. Domains D7 and D8 are allocated to node M4. There is a bias toward restoring the initial allocation scheme when a node becomes available. This bias contributes to faster subsequent query times upon re-entry of a server node due to the use of the data query cache, and routing of subsequent requests to the particular nodes in accordance with this bias.

Detailed Description Text (107):
Referring now to FIG. 33, shown is an example embodiment of a flowchart of method steps for performing a data query. At step 200, a determination is made as to whether a data set in the data query cache corresponds to the current query being made. If so, control proceeds to step 202 where this data is retrieved and used by the query engine in formulating the query results that are displayed to the user. At this point, the processing stops at step 216.

Detailed Description Text (108):
If a determination is made at step 200 that no data set in the data query cache corresponds to the current query being made, control proceeds to step 204 where parents of the data query are determined. In this embodiment, parents of the current query are determined by dropping one of the terms. For example, if the query being made is for "MA AND RESTAURANTS AND FLOWERSHOPS", each of the three terms is sequentially dropped to form all combinations of two possible terms. In this instance, the set of parents is the following:

Detailed Description Text (109):
It should be generally noted that in this embodiment, a search is made for only the parent terms. Similarly, other embodiments may go further in searching for results in the data query cache by also forming grandparent terms, as by dropping two terms. This process can be repeated for any number of terms being dropped and subsequently determining if any data sets in the data query cache correspond to the resulting terms.

Detailed Description Text (110):
At step 205, a determination is made as to whether data results in the data query cache correspond to any of the parent terms. If not, control proceeds to step 212 where a closest ancestor may be used as a basis for starting to form the resulting data set. In one embodiment, preprocessing insures that ancestor-based geography exists. In one implementation, that ancestor is a Verity term list associated with a particular state. This implementation uses API calls to retrieve the data identifiers corresponding to the resulting data to be included in the query results.

Detailed Description Text (111):
If, at step 205, it is determined that there are one or more data sets in the data query cache that correspond to one or more of the parent terms, control proceeds to step 206 where a cost is associated with each parent. One embodiment associates a cost with each parent term in accordance with the number of listings of each parent term. This may also be normalized and used in a percentage form by dividing the number of listings in the parent domain by the total number of listings in the query domain. This percentage represents the probability of a business listing belonging to the parent data set appearing in the database. Control proceeds to step 208 where the parent with the minimum cost is chosen as the starting data set for formulating the data results. At step 210, the minimum cost derivation sequence is applied to produce the resulting data query. Generally, the minimum cost derivation sequence is obtained by operating upon the least probability terms first.

Detailed Description Text (113):
Referring now to FIG. 34, shown is a diagram of one example used in step 210 for determining and applying the best derivation sequence. In this example, the query is for MA AND RESTAURANTS AND FLOWERSHOPS. As represented in state 230, it has been determined that MA is the starting data set which is located in the data query cache. In this example, the parentage has been extended to grandparents, and MA has been determined to be the first ranking data set in terms of parentage and number of listings in the data set. At this point, control proceeds to one of two states, 232 representing "MA AND RESTAURANTS", or 234 representing "MA AND FLOWERSHOPS". The state to which control is advanced depends generally on choosing the path with the minimum associated cost at each step. In this instance, the number of elements in the data sets "FLOWERSHOPS" (state 234) and "RESTAURANTS" (state 232) may be considered in determining cost. If the number of elements in FLOWERSHOPS is less than the number of elements in the data set RESTAURANTS, control proceeds to state 234 where each business listing in the data set FLOWERSHOP is examined to determine if it is also in MA. The resulting data set forms the set of all business listings in MA AND FLOWERSHOPS. In contrast, if the number of elements in the data set RESTAURANTS is less than FLOWERSHOPS, state 232 is entered and similar searching of

the data set is performed. From either state 232 or 234, control proceeds to state 236 where searching of the data set elements is performed to produce the final resulting data set representing "MA AND RESTAURANTS AND FLOWERSHOPS". Generally, the approach just described is to advance to the next state which has the minimum cost associated until the final resulting data set is determined.

Detailed Description Text (114):
It should also be noted that some of the determination of data sets as used in performing queries may be done as preprocessing to partition the data sets. For example, in one embodiment, the data is partitioned by states. The adaptive techniques as described with regard to the GTE Superpages application described herein include partitioning the data sets based on geography, particularly within each state. In this instance, particular server nodes are designated as primary query servers based on geographic location by state. Additionally, as part of this partitioning of requests, the data query caches and term lists of identifiers are also partitioned according to state. In this embodiment, this partitioning is done as a preprocessing step prior to servicing a request in that the identifiers are formed and placed on each dedicated server node. Similarly, other data partitioning may also be performed as part of a preprocessing step. Generally, this partitioning may be determined based on expected data queries and data sets formed accordingly, for example, by examining log files with recorded data query search histories to determine frequently searched categories or combinations of categories.

Detailed Description Text (118):
The Data Query Cache 850, in this embodiment, generally includes a "hot" and "cold" cache. In this embodiment, the caching technique implemented is the LRU (Least Recently Used) policy by which elements of the cache are selected for replacement in accordance with time from last use. These and other policies are generally known to those skilled in the art. Generally, the "hot" cache may include the most recently used items and the cold cache the remaining items. In this embodiment, each of the data query caches and other caching elements as depicted in FIG. 2, may be fast memory access devices, as known to those skilled in the art, used generally for caching.

Detailed Description Text (121):
Data sets that are stored in the data query cache and page cache each correspond to a particular search query. In other words, a mapping technique may be used to map a particular query to corresponding data as stored in the data query cache and the page cache. Generally, this mapping uniquely maps a data query to a name referring to the data set of the data query. In this embodiment, this allows quick access of the data set associated with a particular query and quick determination if such a data set exists, for example, in the data query cache.

Detailed Description Text (122):
Referring now to FIG. 35, shown is a flowchart of an embodiment of the steps for forming a name associated with a data set, as may be stored in the data query cache or page cache. At step 240, a subset of query terms is determined such that a string representing a particular query is uniquely mapped to a name corresponding to a data set. In this embodiment, the subset of keys that are used in mapping a string corresponding to a query to a name of a data set include:

Detailed Description Text (124):
At step 244, a query string corresponding to a particular user query is formed using the original string as formed, for example, by the Parser of FIG. 2. The query string includes only those terms which are included in the subset as identified in step 240. If the original string does not include an item that is in the subset, for example, since the user query does not include the item as a search term, that item is omitted in forming the query string corresponding to the data set. At step 248, this query string is used to determine if a data set is located in the data query cache that corresponds to the current user query request. In this

embodiment, the data sets each correspond to a filename. Thus, a lookup as to whether a data set corresponding to a particular user query exists may be determined by performing a directory lookup, for example, using file system services as may be included in an operating system upon a device which serves as a fast memory access or other caching device.

Detailed Description Text (125):
It should be noted that this technique may be used generally within the Superpages Front End Server and Backoffice to form unique names that correspond to particular search terms. For example, one embodiment may include services for operating upon the original query string as formed by the Parser to produce parents and grandparents of the terms included in a query when performing the method steps of FIGS. 33 and 34 if there is no exact data set match in the data query cache. This may provide the advantage of insulating other code, such as in data encapsulation, from knowing the internal structure of the query string. Generally, as known to those skilled in the art, this is a common programming technique to minimize code portions from changes in data types and structures to minimize, for example, the amount of recompilation when a new data type is introduced or existing data type modified. Other techniques, such as hashing, may be used to generate a unique identifier for the input string, as known to those skilled in the art.

Detailed Description Text (126):
It should be generally noted that a similar mapping technique is used in forming a Page Cache name. The technique used is as described for forming the Query Cache filename with additional qualifying terms in accordance with the "look and feel", such as display features, used to produce the Page Cache name. For example, if the displayed resulting HTML page includes 15 listings/page, the Page Cache name includes a parameter in forming the name uniquely identifying the filename including the result set for a query in this particular display format.

Detailed Description Text (127):
Generally, in this embodiment, the data query cache includes cache objects in which each cache object corresponds to a particular cached query resulting data set. Referring now to FIG. 36, shown is a block diagram of one embodiment of a data set as stored in the data query cache. Generally, each data set 250 includes header information 252 and information corresponding to one or more business listings. Generally, header information may include information describing the data query set, such as the number of business listings in the data set. Other types of information may be included in accordance with each particular application and implementation.

Detailed Description Text (140):
The technique used to store the data in the data cache from memory includes object serialization and deserialization techniques, as known to those of ordinary skill in the art. These techniques transform an internal storage format, as may be stored in random access memory, to a format suitable for persistent storage in a file system, as in the data query cache. The complementary operation is also performed from persistent storage to the in-memory copy. For each of the above-named fields, object serialization, i.e., from memory to persistent storage device in cache, is performed by storing the data type, its length, and the data itself. It should be noted that the length may not be needed for each data field, for example, in fixed length data types. The complementary operation of object deserialization is generally performed by reading the fields in the same order as written to the cache.

Detailed Description Text (142):
It should be noted that in this particular embodiment, the data query cache may include different types of cached geographical data as may be used in performing different data queries. For example, the type of data cached described in the prior paragraphs is the "normal" business listing data as associated with a well-defined

geographic area. Other businesses, for example, such as a florist or an airline, may not be associated with a single well-defined geographic location. A business may not have any geographic bounds, such as if it is an Internet business with a virtual storefront accessible on the Internet. Also, other businesses may be located in a particular well-defined geographic area, such as an airline with a physical presence in a particular city, but the service area which corresponds to the service offered does not correspond to the location of the business itself. To include businesses with these particularities, in addition to the "normal" business listing just described in which the geographic business location and service areas correspond, the concepts of multi-city and total-city placements have been included in this embodiment.

Detailed Description Text (148):
These key-value pairs are extracted from the original query string and appended to the "SCOPE=T" to form the total-city cache name. In one embodiment, these functions of extracting the information from the original query string and forming the total-city cache name may be performed by the same software as forming the name for the data query cache "normal" query name, such as by API calls to the same routines with parameters, as known to those of ordinary skill in the art of programming.

Detailed Description Text (154):
In all the caches, a garbage collection technique may be included to remove or delete cached objects that have been determined to be "old" in accordance with predetermined criteria. For example, in one embodiment using the LRU caching scheme, whenever the amount of free cache space falls below a threshold level, the garbage collection routine is invoked. The threshold level includes parameters relating to a predetermined number of cache objects and the accumulated size of the objects in the cache. In this embodiment, although there may be multiple conceptual caches, such as the "normal" data query cache, the multi-city cache, and the total-city cache, the cached results may physically reside in the same "hot" and "cold" caching devices. However, in this embodiment, the different types of caching results may be accessed independent of the other caching results. Other embodiments may have other organizations of the caches in accordance with other implementation and associated data requirements.

Detailed Description Text (179):
Referring now to FIG. 45, shown is one embodiment of the database included in the Backoffice component as included in FIGS. 2 and 4. Generally, data updates included in the database come from three different sources in this particular embodiment. One source is on-line updates, as provided by users making updates or entering new information for business listing via network connections through the Backoffice component as through the Front End Server. A second source of data updates is based on foreign source updates. Generally, foreign source updates are those update records which come from a different data source than the original existing database. A third type of data integration or update source is referred to as a native source update. Generally, a native source update is when an updated version of the existing database having the same source as the existing database is provided. For example, a database copy may be provided as an update on a monthly basis using full sets of data where a data provider provides an updated version of the same data set. The native source data integration procedure integrates those changes in the new data set into the existing database. This is in contrast to a foreign source update, for example, where the existing database is provided by one vendor, and the update records for example, are provided by a different vendor. The update vendors being from a foreign source are called foreign source data integration or updates.

Detailed Description Text (236):
Generally, the text data included in this data transfer may be characterized as structured data, as included in text which is displayed to the user. The second type of data generally transferred is denoted as "blob" data which is generally not

able to be decomposed or operated upon in different portions. For example, blob data may include a machine-executable program which is generally binary data type. Generally, the technique uses two separate data channels in which each channel transfers a different type of data. In this particular embodiment, one data channel is used to transfer the text data, and Database Link.TM. software, as included in the commercially available Oracle.TM. database, is used to facilitate database communication of text data. Therefore the database routines, such as those included in the Database Link software, may be used in transferring text data between databases. In this particular embodiment, the Oracle database does not support direct non-text manipulation, such as for transferring data of different types, such as blob data. Therefore, a second different data channel is used to transfer the blob data from one database to another in which the second channel is external to the database since the version of the Oracle database software used in this embodiment does not provide the needed support for direct non-text data manipulation. The blob data, which may also generally be characterized as multi-media data, is transferred asynchronously from the text data between databases.

Detailed Description Text (258):
The overall technique is generally to copy the text and blob or multi-media data asynchronously on two separate channels. This data is copied from a first database to a second database. Initially, the data is located on the second database in a temporary location until all of the portions of the data associated with a particular data transfer arrive at the second database. When it has been determined that all portions of the data have successfully arrived on the second database, the assembly process of copying the data from the temporary locations and merging the information into other data tables is performed on the second database.

Detailed Description Text (260):
The data transfer technique described is generally a technique for transferring data using two data links between two databases. One of these data links is an internal data link with respect to the database, the second data link is an external data link with respect to the database. The internal data link is optimized for the structured text data transfer while the external one is optimized for the multimedia data transfer, such as the transference of data stored in binary objects in the database. This technique for data transfer generally alleviates the limitations of the existing database technology which does not provide for the transferring of multimedia objects using the internal data link. Moreover, by using the two data links to transfer the various data types, performance and stability are improved over an alternative prior art approach which uses only the external link for transferring both text and multimedia or blob data.

Detailed Description Text (269):
Once the data modifications are incorporated into the Backoffice component, the data updates, including the updates to advertisement data and other data associated with each business listing, may be propagated to the Front End Server component. The non-text or multimedia data, for example, as included in advertisements with image files, may be transferred to the Front End Server from the Backoffice using multimedia transfer techniques, as generally described in other sections of this description. The updates to the Primary Database included in the Front End Server may be communicated as a table of commands created in the Backoffice component and transferred, as by a network connection, to the Front End Server. Generally, in this embodiment, the table created in the Backoffice includes an application developed command language corresponding to the various types of record updates and modifications that may be included in this particular embodiment. Each of these commands may be further translated in the Front End Server into one or more actual database commands that perform the table operation. For example, an entry in the table of database update commands may be specified as follows:

Detailed Description Text (287):
Also, in this particular embodiment, the contents of the Page Cache 848 and the

Query Cache 850 are reinitialized when an update is performed, as in performing the incremental update procedures described in conjunction with FIGS. 31 and 32. The data included in the PHTML execution tree is also reinitialized.

Current US Class (1):
707

Previous Doc Next Doc Go to Doc#